## I. Implementation and Performance Evaluation

### A. Implementation Details

We implemented a prototype of our scheme[1] in Python and Solidity v0.8.17 [1] and conduct experiments on a computer with an Intel(R) Core(TM) i3-8130U CPU, 8 GB of RAM. The experiments are conducted on the Ethereum test network Ganache [2]. Similar to [3], we instantiate ADS by storing blocks via transaction on Ethereum. We interact with the Ethereum Network using the web3.py [4] API package. The **Init** method of ADS is instantiated by creating a public address and private key of a Ethereum wallet, while **Put** and **Get** are executed by calling the Ethereum APIs $web3.eth.send\_transaction()$ and $web3.eth.get\_transaction()$, respectively. We use the PyCryptodome [5] library to implement cryptographic primitives. In our implementation, PRFs and symmetric encryption are implemented with AES-256, hash functions are implemented with SHA-256. For the digital signature $\Pi_{\text{sig}}$, we use the Ethereum API $web3.eth.account.sign\_message()$ and the Solidity function $ecrecover()$. For $KC$ and $Query$, we programmed two smart contracts using the Solidity language and deploy them to the Ganache test network. We use the solcx [6] package for compiling Solidity.

For the experiments, we use a real-world email dataset [7] to evaluate the performance of our prototype. We extracted subsets of emails from the original dataset as document collections and extracted keywords for each document to get different sizes of document/keyword pair collections. Our evaluations and experiments aim to assess the practicality of our scheme, including storage overhead, Ethereum gas consumption, on-chain query and verify latency.

TABLE I
Storage Overhead

| Scheme | | Number of keyword-document pairs | | |
|---|---|---|---|---|
| | | $2.5 \times 10^3$ | $5 \times 10^3$ | $1 \times 10^4$ |
| Guo *et al.*' | Client | 47.27KB | 110.90KB | 236.13KB |
| | Blockchain | 67.28KB | 138.64KB | 295.16KB |
| | Server | 229.13KB | 458.27KB | 916.53KB |
| Ours | Blockchain | 132.38KB | 264.76KB | 529.52KB |

### B. Storage Overhead Evaluation

On the client side, our scheme only requires the client to store secret keys, which is $O(1)$ in all cases, reducing the client storage to a minimum. For storage on the blockchain, if the total number of document/keyword pairs that are historically added and deleted from the database is $N$, the storage space on the blockchain is at worst $O(N)$. To gain a better understanding of the performance, we compare the storage overhead of our design with the design by Guo *et al.* [8][2], where the client maintains keyword and query state, the server stores an encrypted index, and only digests are stored on-chain for result verification. Table I. shows the storage overhead of our design compared to the scheme by Guo *et al.* [8] over the number of keyword-document pairs ranging from 2.5 K to 10 K. Results in Table I confirms that our scheme achieves minimum storage overhead for the client.

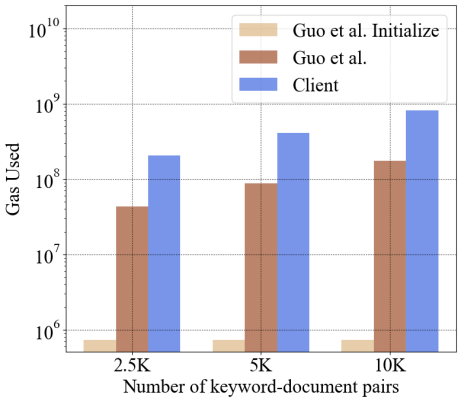


Fig. 1. Gas for update.

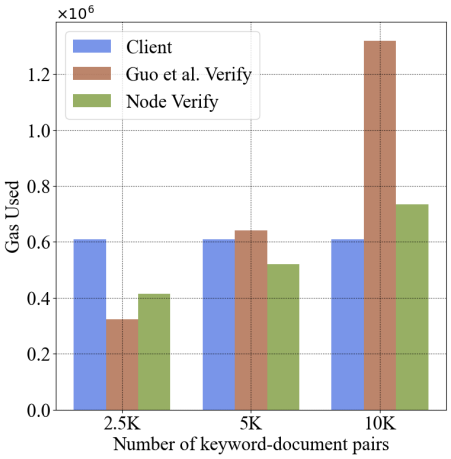


Fig. 2. Gas for search and verify.

### C. Cost Evaluation

The cost of storage and computation on Ethereum are measured by gas. The total cost of Ether is calculated by $gasCost \times gasPrice$. In our experiment setting, the gas price is set to 20 Gwei, where 1 Gwei is $10^{-9}$ Ether. The gas used by the client includes the gas for deploying smart contracts and sending transactions, and for the node, it is the gas for smart contract function execution. We measured the gas used in each phase by the client and the node as shown in Fig. 1 and Fig. 2. For a better understanding of the gas consumption, we also include the gas used in the scheme by Guo *et al.* [8] for comparison.

[1]https://github.com/xuanchenunimelb/FIT4441Project

[2]There are other similar blockchain-based VDSSE schemes. However, only the source code of this one is publicly available and works well on our testbed.

In the update phase, the client uploads an encrypted message to the blockchain, causing linearly growing gas usage with the number of updates of keyword-document pairs. The scheme by Guo *et al.* [8] actually adopts off-chain storage. Their scheme only uploads digests to the blockchain for verification. As Fig. 1 shows, our on-chain design has a relatively higher cost. This is a tradeoff to avoid single-point failure.

Fig. 2 shows the gas used for queries. The search phase in our design requires the client to deploy a smart contract, which requires a constant number of gas. For the node, execution of the smart contract function consumes gas. In Guo *et al.* 's work, the verification is also executed with a smart contract, which also consumes gas. In Fig. 2 we can see the verification of our scheme consumes less gas than Guo *et al.* 's work when the database has more than 2.5 K keyword-document pairs.

TABLE II
SEARCH AND VERIFY LATENCY

| **Scheme** | **Number of keyword-document pairs** | | |
|---|---|---|---|
| | $2.5 \times 10^3$ | $5 \times 10^3$ | $1 \times 10^4$ |
| Guo *et al.*' search | 0.13s | 0.33s | 0.87s |
| Guo *et al.*' verify | 1.28s | 2.06s | 2.59s |
| Ours search | 5.92s | 18.65s | 63.58s |
| Ours verify | 1.78s | 2.36s | 2.97s |

### *D. Query and Verify Latency Evaluation*

We also measured the query latency of our scheme. We again compare the performance with the scheme by Guo *et al.* [8], where the search is performed by the server off-chain and later verifies on-chain. The result is shown in Table II. The result shows that the verification time is very close to Guo *et al.* [8], yet the search time of our design is significantly higher. This is mainly because the off-chain search is more efficient than the on-chain search. We will make efforts to improve the performance of the on-chain search by integrating multiple update records into one transaction so as to reduce the database accesses.

## REFERENCES


[1] "Solidity documentation," https://docs.soliditylang.org/en/v0.8.17/index.html, 2023.
[2] "Ganache," https://trufflesuite.com/docs/ganache/, 2023.
[3] D. Adkins, A. Agarwal, S. Kamara, and T. Moataz, "Encrypted blockchain databases," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 241–254.
[4] "web3.py documentation," https://web3py.readthedocs.io/en/stable/web3.eth.html, 2023.
[5] "Pycryptodome documentation," https://pycryptodome.readthedocs.io/en/latest/, 2023.
[6] "py-solc-x documentation," https://solcx.readthedocs.io/en/stable/.
[7] "Enron email dataset," https://www.cs.cmu.edu/enron/.
[8] Y. Guo, C. Zhang, C. Wang, and X. Jia, "Towards public verifiable and forward-privacy encrypted search by using blockchain," *IEEE Transactions on Dependable and Secure Computing*, 2022.